AFRL-IF-RS-TR-2006-33
Final Technical Report
February 2006

DETECTOR AND EXTRACTOR OF FILEPRINTS (DEF) EXTENSION

Black River Systems Company, Inc.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

APPROVED: /s/

ANDREW NOGA
Project Engineer

FOR THE DIRECTOR: /s/

JOSEPH CAMERA
Chief, Information and Intelligence Exploitation Division
Information Directorate

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 074-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503			
1. AGENCY USE ONLY (Leave blank)	**2. REPORT DATE** FEBRUARY 2006	**3. REPORT TYPE AND DATES COVERED** Final Apr 2004 – Sep 2005	
4. TITLE AND SUBTITLE DETECTOR AND EXTRACTOR OF FILEPRINTS (DEF) EXTENSION			**5. FUNDING NUMBERS** C - FA8750-04-C-0207 PE - 63789F PR - STG3 TA - 04 WU - 09
6. AUTHOR(S) Richard Henry			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Black River Systems Company, Inc. 162 Genesee Street Utica New York 13502			**8. PERFORMING ORGANIZATION REPORT NUMBER** N/A
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/IFEC 525 Brooks Road Rome New York 13441-4505			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER** AFRL-IF-RS-TR-2006-33
11. SUPPLEMENTARY NOTES AFRL Project Engineer: Andrew Noga/IFEC/(315) 330-2270/ Andrew.Noga @rl.af.mil			
12a. DISTRIBUTION / AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED			**12b. DISTRIBUTION CODE**

13. ABSTRACT *(Maximum 200 Words)*

This technical report summarizes the work performed under the Detector and Extractor of Fileprints, DEF, Extension contract for the Air Force Research Laboratory, Rome Research Site, Information Directorate. The focus of the underlying effort was comprised of an improved file tamper protection scheme, new data stream MIME-type identification techniques and improving a portable Java implementation designed to demonstrate that functionality. The existing tamper detection capabilities were enhanced to allow for greater localization of tampering while minimizing storage requirements for the created outputs. Data stream MIME-type identification techniques were expanded to include the use of new statistical DEF measurements and Bayesian classifiers. The report provides conclusions and recommendations on the development, implementation and enhancement of new and existing data integrity and data forensics features using the DEF algorithm.

14. SUBJECT TERMS Detector and Extractor of Fileprints (DEF) Extension DEFE, tamper detection, data forensics, data integrity, Bayesian			**15. NUMBER OF PAGES** 33 **16. PRICE CODE**
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	**18. SECURITY CLASSIFICATION OF THIS PAGE** UNCLASSIFIED	**19. SECURITY CLASSIFICATION OF ABSTRACT** UNCLASSIFIED	**20. LIMITATION OF ABSTRACT** UL

Table of Contents

List of Figures

List of Tables

1. Summary

This is the final report for contract FA8750-04-C-0207, entitled “Detector and Extractor of Fileprints (DEF) Extensions”. The goal of this effort was to develop a multi-platform software application that demonstrates the potential and effectiveness of applying the DEF algorithm to two related problems: 1) Determining the MIME file type of an unknown data stream, and 2) Determining if an arbitrary data file has been modified and/or tampered. The DEF algorithm was developed by the Air Force Research Laboratory (AFRL). This effort is an extension to, and builds on the product of, an earlier effort entitled Detector and Extractor of File-prints (DEF) Prototype [1], contract F30602-03-C-0068.

2. Overview

The DEF algorithm is a technique for characterizing arbitrary data sequences. DEF is a periodogram based process and utilizes the Adjustable Bandwidth Concept (ABC) [2, 3] detector to generate a unique file-print in a condensed form. Generation of the file-print is a repeatable process based on a secure parameter key. In this way a data sequence can be characterized and its file-print recorded for subsequent use in determining whether the file was modified. The ABC algorithm has been developed and patented by Dr. Andrew Noga at AFRL.

Given the secure key, the repeatable process can generate a test file-print set to be compared with the original set. Variations in the candidate file-prints indicate that the file has been modified. Only the secure key needs to be protected to maintain the integrity of the system. The original file and the data file-print can be publicly shared.

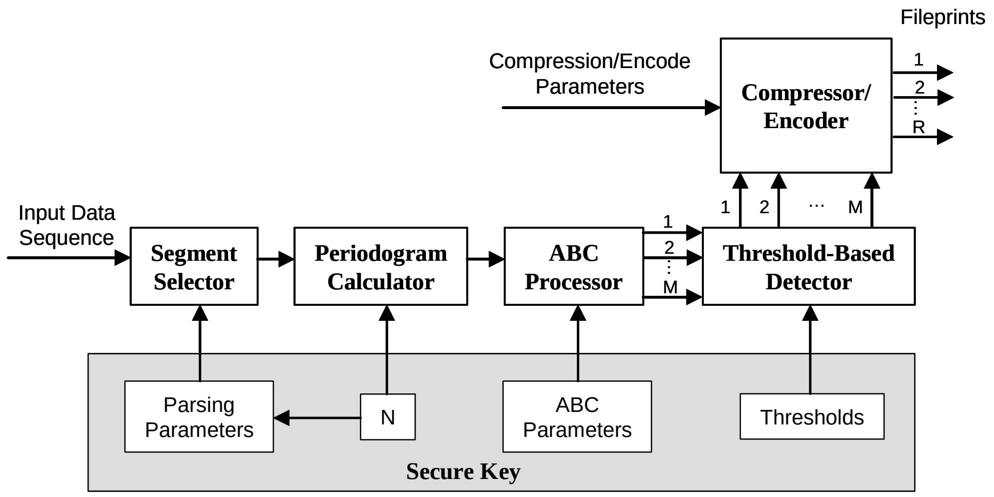


Figure 1 - The DEF Algorithm

The input data sequence is run through a segment selection process, which is manipulated by a pseudo-random number sequence seeded by the provided secure key (see Figure 1). After the segment has been selected, a periodogram is calculated for the selected segment and the output of that process is provided to the ABC detector. A threshold is applied to that output and the resulting data is encoded to the DEF file-prints. The process is repeated for each segment in a file to create a complete file-print.

While DEF was originally intended primarily for file-print generation (and thereby for data tamper detection), statistical evidence has suggested that a byproduct of the process is a relationship between the structure of the file-print and that subjects MIME-type. One of the goals of this effort was to exploit this relationship, using it to recognize the MIME-type of an unknown data sequence.

2.1 Data Tamper Detection

2.1.1 Detecting Tampering in Files and Data

At the core of the varied fields of computing you will invariably find data. These streams of data, and more specifically files, found on modern computer systems are there to serve a purpose. As these purposes become more varied and the data increases in its importance to society, the desire to influence that data and more specifically the decisions that ensue upon use of that data also increases. Data that is altered improperly in this way is considered to be the target of tampering. Some of the more common examples of data tampering include computer viruses, intentional misconfiguration of services and security policies for networked systems, and covert image manipulation.

There are a number of goals represented by the term ‘tamper detection’, but most of them can be grouped into a short list of increasingly more specific and difficult tasks:

- Identify that a change has occurred.
- Localize the change to specific locations in the data.
- Make inferences into the goal of the changes.

The DEF tamper detection implementation attempts to deal with the first two tasks, to first detect a change, and second to be able to derive what data has been changed. Beyond those two basic goals the DEF implementation also provides us with some added benefits.

The file-print generation process changes predictably based on the private integer key applied, as described in the algorithm overview. This can be leveraged as a form of security for the process, making it more difficult for external parties to produce substitute outputs in scenarios in which the outputs are delivered with the subject data. One limitation of this process is that if the integer key is changed our ability to accurately localize changes is diminished.

In many cases the size of the DEF outputs required to supply the necessary information for the tamper detection process is much less than that of the original file, providing some level of compression unlike a direct comparison of the original file to a backup copy.

2.1.2 A Few Contemporary Techniques

There are two applicable contemporary techniques that we will examine to add points of reference to the discussion of DEF as a data tampering detection system.

2.1.2.1 Watermarking

When a file is digitally watermarked, information about the content, author, copyright owner or other related facts are encoded into the file with the characteristic that it does not overtly change the meaning of the content of the file.

This watermark may be easily detectable, possibly even visibly in images, but often it is encoded in a way in which day-to-day users of the content would not notice its presence. From a file tamper detection perspective this technique would serve to identify known modified content as being a derivative of the original and prevent other agents from producing a forgery.

2.1.2.2 Hashes

Modern hashing algorithms work much like and are often derived from modern cryptography techniques with the key difference being that the process is not reversible. These predictable one-way cryptographic algorithms are known as digests. They are normally used to provide information integrity assurance in situations where a file is transmitted over public networks or stored on unsecured servers such as file mirroring sites.

The benefits of using digest algorithms is that their outputs are always the same size, are very small and are sensitive to even the most insignificant changes in the subject data stream.

The hashes provided by digest algorithms have no method for discerning where data has been changed, only that the data sequence in its entirety is or is not intact. Normally files are distributed to a number of sites for download and hashes are stored on the primary site. If the user wishes to verify the content of a mirrored version of an archive, they get the original hash from the primary server and then generate a hash on the mirrored file on their local system and then compare. In contrast, the DEF algorithm allows for a degree of localization of the tampering within a file, for a modest increase in required data storage.

2.1.3 Applying the DEF Algorithm to Tamper Detection

As each segment is processed through the DEF algorithm, a periodogram is calculated. In our implementation we used a Fast Fourier Transform (FFT), which outputs the transformed data sequence and a mean power value. This mean power term for the segment is well suited for tamper detection since even the slightest changes in the data sequence will be reflected in its value.

For each segment processed by DEF, the mean power term (L1 threshold value) is written to a threshold file. This threshold file alone provides enough information to detect changes in the file and provide relatively detailed locations for those changes. Given the bits/segment n, the file size in bytes f and the DEF processing block size b, we can define the storage function for this level 1 threshold file:

$$\frac{n\left\lceil \frac{f}{b} \right\rceil}{8} \qquad \textbf{2-1}$$

Initially implementations used only this primary threshold file but more detailed location abilities were possible. We found that if you transposed the incoming data stream segments and created the FFT mean power terms in this transposed dimension (L2 threshold value) we could provide even more localization. The only caveat to this approach is that it is not efficient for files that are smaller than ~8Kb. Given the bits/segment n, the file size in bytes f and the DEF processing block size b, we can define the storage function for the level 2 threshold file:

$$\frac{nb\left\lceil \frac{\left\lceil \frac{f}{b} \right\rceil}{b} \right\rceil}{8} \qquad \textbf{2-2}$$

When combined these two functions provide the storage function for both level 1 and 2 threshold files, again given the bits/segment n, the file size in bytes f and the DEF processing block size b:

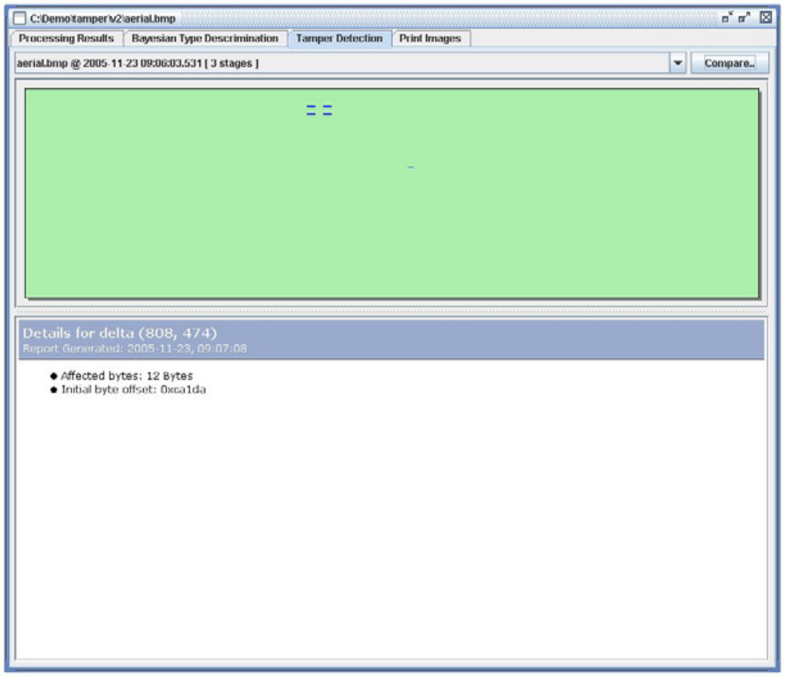
$$\frac{n\left(\left\lceil \frac{f}{b} \right\rceil + b\left\lceil \frac{\left\lceil \frac{f}{b} \right\rceil}{b} \right\rceil\right)}{8} \qquad \textbf{2-3}$$

If only the level 1 thresholds are generated, the process creates no tangible overhead on the DEF processing system as the FFT is executed as a function of generating the DEF prints for the data. The level 2 thresholds require more transforms to be executed and will take the user significantly more time to generate.

Once we have generated a set of threshold files for a known data stream and the comparison candidate data stream, we compare the threshold files for differences. In the case of a L1-only comparison we are able to localize changes to within any 1Kb block of the subject data stream. If we generate the L2 thresholds as well we are given a best case scenario with byte level accuracy and a worst case scenario equal to the L1-only scenario.

The reason for this is that the L1 and L2 threshold files are generated independently and are then correlated. With a threshold processing block we can not know which L1 entities relate to which L2 entities. We must therefore correlate every L1 entity to every L2 entity in related processing blocks. After we have correlated the L1 and L2 entities we can translate these pairs back into a corresponding byte-offset in the data stream and mark them as modified.

After this comparison is made the user will be presented with the report shown in Figure 2, unless they were processing a 24 bpp bitmap image, in which case they will be provided with a visualization as shown in Figure 3.



aerial.bmp @ 2005-11-23 09:06:03.531 [3 stages]

Details for delta (808, 474)
Report Generated: 2005-11-23, 09:07:08

- Affected bytes: 12 Bytes
- Initial byte offset: 0xca1da

Figure 2 - DEF Tamper Results, General Report

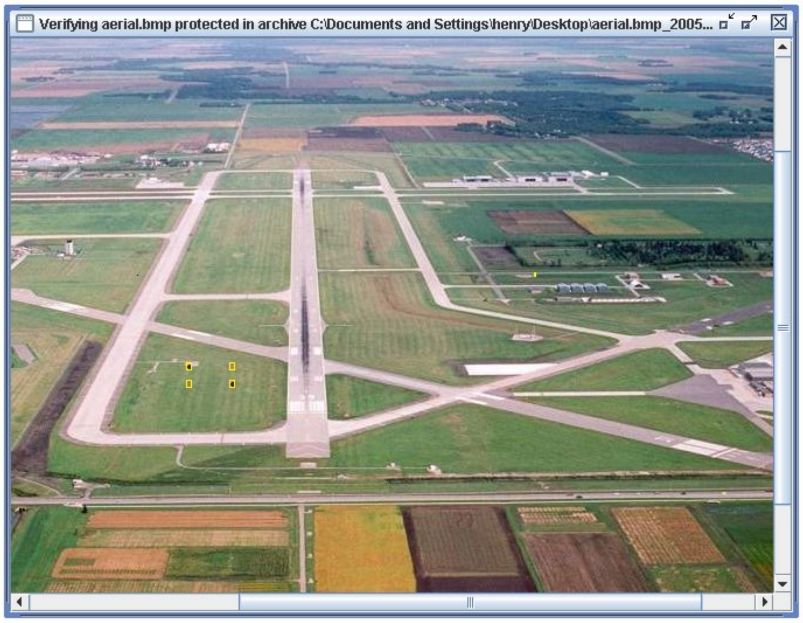

Verifying aerial.bmp protected in archive C:\Documents and Settings\henry\Desktop\aerial.bmp_2005...

Figure 3 - DEF Tamper Results, BMP Image

2.2 Data Stream Identification

When the DEF algorithm is executed on arbitrary data it produces a binary data representation of that sequence. We are looking to identify and quantify statistically significant numerical ‘features’ that binary output in a way that will allow us to use probability to make inferences into the data that generated that sequence. The potential features evaluated for data stream identification are listed and described below.

2.2.1 Average Band Occupancy

Given a data source with L segments for processing, each of which creates these binary sequences b with length in bits N and using the feature size in bits S to provide the associated band occupancy B gives:

$$B_a = \frac{\sum_{i=1}^{L}\left(\sum_{j=1}^{S} b^{i}_{a\times S+j}\right)}{L}, \qquad \textbf{2-4}$$

where a is a number between 0 and (S/N-1).

We use these band occupancies as a set of observable values, or a feature vector, by which our data sequences can then be evaluated using simple statistics and probability theory. Baseline procedures included 3 stages of processing and in general practice a 1024 point FFT was processed using the ABC detector, providing us with a 512 bit binary string for each of those stages. A value of 32 bits was selected for the feature size, chosen with the goals of providing us with chunks of data that are easy to manipulate programmatically and that are large enough to express potentially generated features effectively. This provided us with 16 base metrics per stage, 48 in total.

These core DEF measurements, or features, also provided the basis for complex features. These complex features attempt to measure relationships between features in the core set.

2.2.2 Average Band Ratio

The average band ratio represents the number of bits on in a specific feature divided by the number of bits on in the following feature averaged over all the features in that position over the entire file:

$$A_a = \frac{\sum_{i=1}^{L}\left(\frac{\sum_{j=1}^{S} b^i_{a\times S+j}}{\sum_{j=1}^{S} b^i_{(a+1)\times S+j}}\right)}{L}, \qquad \textbf{2-5}$$

where a is a number between 0 and (S/N-2).

2.2.3 Average Band XOR

The average band XOR represents how much the content of these bands changes from feature to feature:

$$X_a = \frac{\sum_{i=1}^{L}\left(\sum_{j=1}^{S} b^i_{a\times S+j} \oplus \sum_{j=1}^{S} b^i_{(a+1)\times S+j}\right)}{L}, \qquad \textbf{2-6}$$

where a is a number between 0 and (S/N-2).

2.2.4 Variance

The variance of the band occupancy is given by:

$$S_a^2 = \frac{\sum_{i=1}^{L}\left(\sum_{j=1}^{S} b^i_{a\times S+j}\right)^2}{L} - B_a^2 , \qquad \textbf{2-7}$$

where a is a number between 0 and S/N.

2.2.5 Standard Deviation

The standard deviation of the band occupancy is given by:

$$S_a = \frac{\sqrt{S_a^2}}{B_a}, \qquad \textbf{2-8}$$

where a is a number between 0 and S/N.

2.2.6 Entropy

The entropy of a data file was identified as a potential additional ‘feature’ for data stream identification. The entropy of a continuous random *f(x)* variable information is defined as

Continuous Entropy $$S = -\int_{-\infty}^{\infty} f(x)\log f(x)dx\,, \qquad \textbf{2-9}$$

For discrete data sequences, Shannon [4] developed the concept of discrete entropy.

Shannon's Entropy (bits) $$S = -\sum_{i=1}^{n} p_i \log_2 p_i\,, \qquad \textbf{2-10}$$

Shannon's Entropy (nats) $$S = -\sum_{i=1}^{n} p_i \ln p_i\,, \qquad \textbf{2-11}$$

In each case, *S* denotes the entropy of the data sequence, while p_i denotes the probability of occurrence of each byte in the sequence. Choice of log bases in the calculation defines the units of entropy, but does not materially affect the calculation otherwise.

While Shannon and continuous entropy definitions are well-known, Tsallis [5] has more recently introduced the concept of non-extensive entropy. Nonextensive entropy includes the factor *q*, which is experimentally selected to best fit the data source.

Tsallis Nonextensive q-Entropy $$S_q = \frac{1-\sum_{i=1}^{n} p_i^q}{q-1} \qquad \textbf{2-12}$$

In this effort, nonextensive entropy was calculated for the test data set with several trial values for q, with magnitudes between -2 and +2.

2.3 Acquiring Sample Data

The sample population used to create the tests used in the Bayesian classifier has a lot to do with the goals of the classifier itself. In our work we wanted to represent the MIME-types as accurately as possible so we used random files collected from the internet.

Random words were selected from the English dictionary and web searches were executed using those terms. Links to those files were collected from the search results and then downloaded using the GNU wget application.

We then developed scripts to remove mislabeled, broken or otherwise unacceptable files from the collected samples. Those files were then broken into groups ranging in size from 500 to 1000. In most cases 1000 samples contained enough information to properly describe the features we were interested in studying.

2.4 Analyzing DEF Features with Probability

To find similarities and differences in the DEF Features between file types, we needed to obtain a large sample of files. How to sample a random set of files is a difficult problem that changes based on what the researcher actually wants to achieve. For example, if we wish to obtain samples that properly represent user created content such as images and word processing documents, a sample set consisting of all of the files on the average computer hard disk will likely not be a good sample since it will contain a large number of raw data and executable files for software installed on the system.

To represent such a population properly you need a source that is, largely if not entirely, comprised of such "user generated" files. Fortunately the internet provides us with such a sample set, much of its content being comprised of the desired documents and images. Picking random words from an American English dictionary and then using those words to search the internet provides us with a number of resulting resources. We filter this resource list to include only files of the types for the desired MIME-type. After downloading the individual files we must test them to ensure that they are valid files of their marked type.

We used this system to acquire 6 test datasets comprised of 500 to 1000 files of each of our target MIME-types and accept these samples as representing the total desired file population. DEF Features are calculated in these type groupings giving us a matrix for each type made up of the individual DEF feature vectors. We then look at each one of the features across the samples for each type and create a relative frequency histogram, generating a *type count x feature count* atrix of histograms as shown in Figure 4.

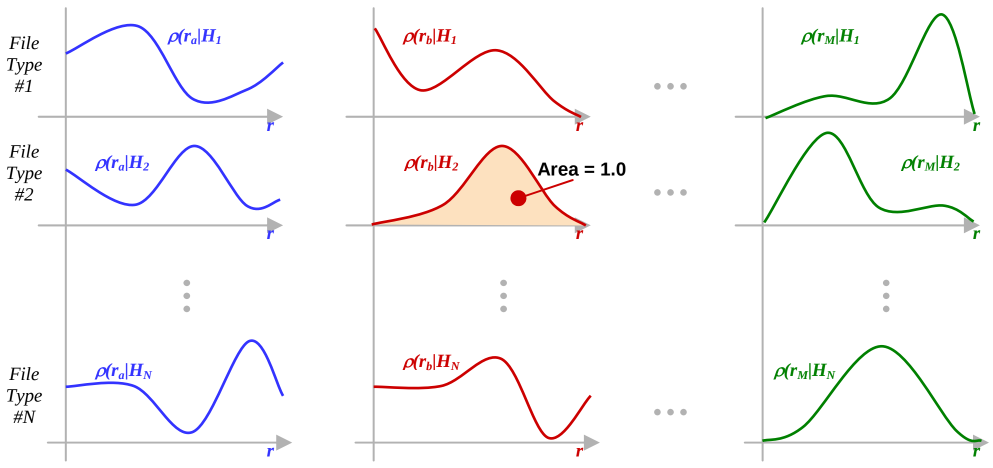


Figure 4 - Relative Frequency Histograms for DEF Features

We interpret this histogram, with the help of the Law of Large Numbers, as a probability density function providing the possibility that a specific value will occur in a specific feature in a sample of a specific MIME-type. We then collect unknown samples, generate the same features and look up the probabilities of those occurring for each MIME-type, or hypothesis i, given each feature which we interpret as the intersection of these random variable sets, r.

$$P(H_i \mid \vec{R}) = P(H_i \mid r_a, r_b, \ldots, r_M) = P(H_i \mid r_a \cap r_b \cap \ldots \cap r_M) \quad \textbf{2-13}$$

The conditional probability of hypothesis j given the observation vector, R, is defined as the intersection of the hypothesis event with the event that the specific observation vector occurs, normalized by the probability of R, the observation event.

$$P(H_j \mid \vec{R}) = \frac{P(\vec{R} \cap H_j)}{P(\vec{R})} \quad \textbf{2-14}$$

Noting that we can express the intersection of two conditional events with two different expressions, we can write a common form of Bayes Rule.

$$P(H_j \mid \vec{R}) = \frac{P(\vec{R} \mid H_j) \cdot P(H_j)}{P(\vec{R})} \qquad \textbf{2-15}$$

Next, we can express the total probability in the denominator as a linear combination of mutually exclusive event probabilities.

$$P(r_a \cap r_b \cap \ldots \cap r_M) = P(\vec{R}) = \sum_{i=1}^{N} P(\vec{R} \mid H_i) \cdot P(H_i) \qquad \textbf{2-16}$$

Substituting this expression into the denominator, we have our expression for the probability of Hypothesis j given the feature vector R,

$$P(H_j \mid \vec{R}) = \frac{P(\vec{R} \mid H_j) \cdot P(H_j)}{\sum_{i=1}^{N} P(\vec{R} \mid H_i) \cdot P(H_i)}, \qquad \textbf{2-17}$$

and using the definitions of probability distribution and probability density we can show that we can substitute density functions for the probabilities in the solution,

$$P(H_j \mid \vec{R}) = \frac{\rho(\vec{R} \mid H_j) \cdot P(H_j)}{\sum_{i=1}^{N} \rho(\vec{R} \mid H_i) \cdot P(H_i)}. \qquad \textbf{2-18}$$

We assume that all MIME-types, or hypothesis, are mutually exclusive and equally likely therefore,

$$P(H_i) = P(H_j), \qquad \textbf{2-19}$$

$$\sum_{i=1}^{N} P(H_i) = 1.0, \qquad \textbf{2-20}$$

$$P(H_i) \sum_{i=1}^{N} (1) = 1.0, \qquad \textbf{2-21}$$

$$P(H_i) = \frac{1}{N}. \qquad \textbf{2-22}$$

This information allows us to simplify our calculation of the probability of a hypothesis given such a feature vector to the following,

$$P(H_j \mid \vec{R}) = \frac{\rho(\vec{R} \mid H_j)}{\sum_{i=1}^{N} \rho(\vec{R} \mid H_i)}. \qquad \textbf{2-23}$$

2.5 Finding Effective Features

Once the statistics for the features described in section 3.1 are created for all of the types of interest, we have to decide which tests will provide the most effective discrimination results. Also we have to decide on whether tests, once selected, are statistically independent of each other.

While building our Bayesian classifier, we started with the test which, when compared across all types, shared the least amount of density in their histograms. Once a test was selected that test was checked against the remaining tests for correlation. A threshold for acceptable correlation was applied and those tests were excluded from future consideration.

There is still much work to be done in the areas of feature generation and selection. More specifically, the test selection process used did not take into account how effective entire chains of tests were. If close decisions are made at one stage in the process, strong tests could be eliminated by the correlation threshold that would have been beneficial to the outcome of the discrimination. Also the tests we used were often based on similar properties of the data, such as variance, and a less uniform test bed could plausibly yield more effective results.

2.6 DEF Classification Performance

The base types used in the DEF classification process included:

- Bitmap Images
- Microsoft Word Documents
- Microsoft Excel Documents
- Microsoft PowerPoint Documents
- PNG Images
- GIF Images
- JPEG Images
- Portable Document Format Files

The following confusion matrices represent how the algorithm performed on sets of data of each MIME-type. The rows represent what MIME-type is being processed and are not directly related to any other row. The columns show where the detections in that group fell as a percentage of the whole. Following each detailed matrix is a matrix simplifying the results less specific, but more functional, groups.

	BMP	**DOC**	**XLS**	**PPT**	**PNG**	**GIF**	**JPG**	**PDF**
BMP	**97.60**	0.20	2.20	0.00	0.00	0.00	0.00	0.00
DOC	1.40	**80.40**	3.60	11.40	0.60	0.00	1.00	1.60
XLS	2.61	14.83	**78.36**	2.81	0.60	0.00	0.00	0.80
PPT	0.60	8.63	2.61	**64.66**	1.81	0.40	5.62	15.66
PNG	0.60	0.00	0.00	0.40	**63.65**	8.03	24.30	3.01
GIF	0.81	0.20	0.20	0.20	7.47	**84.65**	4.65	1.82
JPG	1.20	1.00	0.00	5.01	11.62	7.01	**66.73**	7.41
PDF	0.20	2.40	4.20	6.20	1.60	0.40	9.00	**76.00**

Table 1 – Confusion Matrix, General Classifier

	Office	**CompImg**	**Other**
DOC	**95.40**	1.60	3.00
XLS	**96.00**	0.60	3.41
PPT	**75.90**	7.83	16.26
PNG	0.40	**87.98**	3.61
GIF	0.60	**96.77**	2.63
JPG	6.01	**85.36**	8.61
PDF	12.80	11.00	**76.20**
BMP	2.40	0.00	**97.60**

Table 2 – Confusion Matrix, General Classifier Simplified Results

The most troublesome combinations were PowerPoint vs. PDF and JPEG vs. PNG. In the first case those types share the combination of textual and graphical data. PNG and JPEG images both provide a very high level of compression, and as a side-result, randomization, which could provide some insight as to why they appear so similar.

The process we employed is statistical in nature so we took a look at some of the basic statistical properties of the JPEG and PNG file types. In all test cases the two highly compressed image types were nearly indistinguishable, as shown in Figure 5.

We also examined the measures of entropy described in section 3.2.6 to check for statistically significant variation between the PNG and JPEG types. Discrete and Tsallis q-entropy results are shown in Figure 6 and Figure 7 respectively.

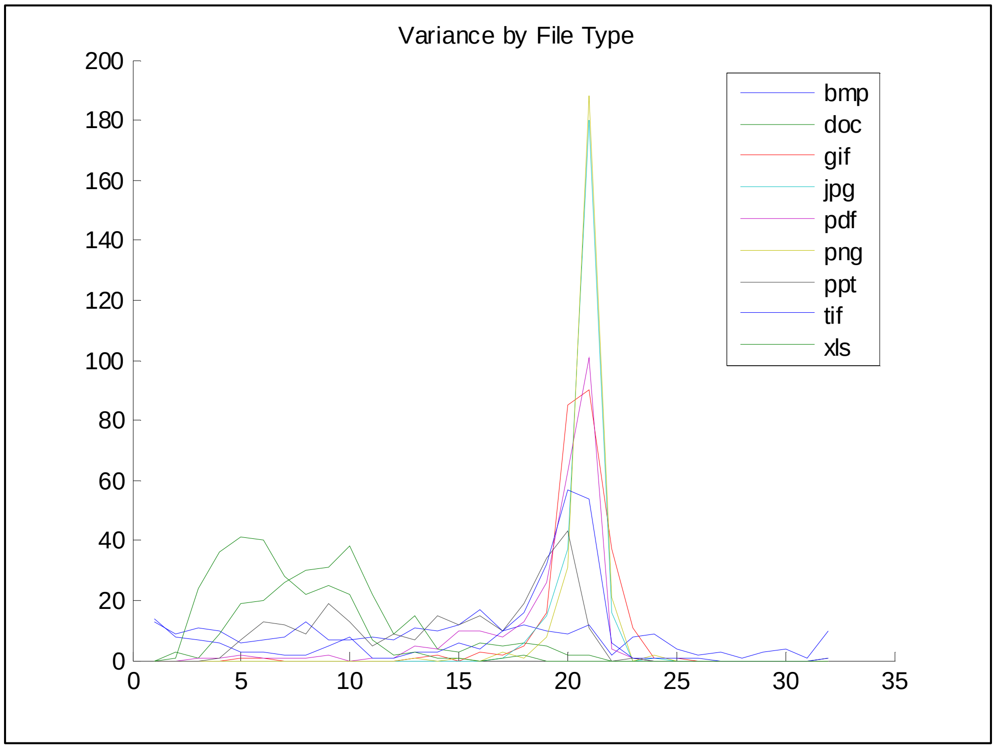


Figure 5 - Variance by MIME-type

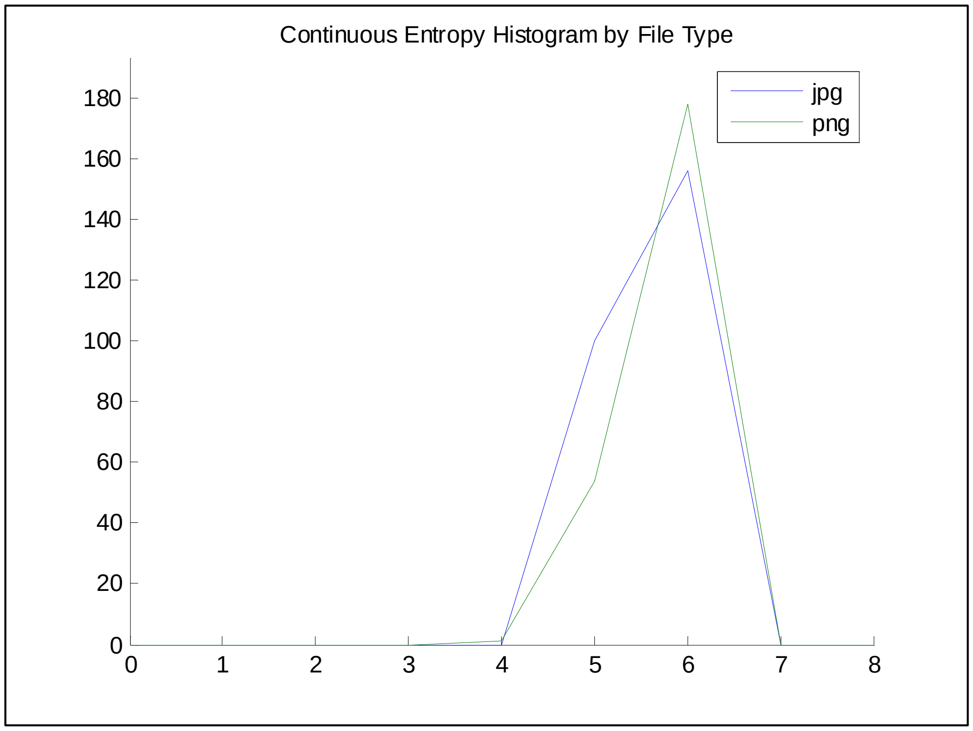


Figure 6 - Discrete Entropy for PNG & JPEG

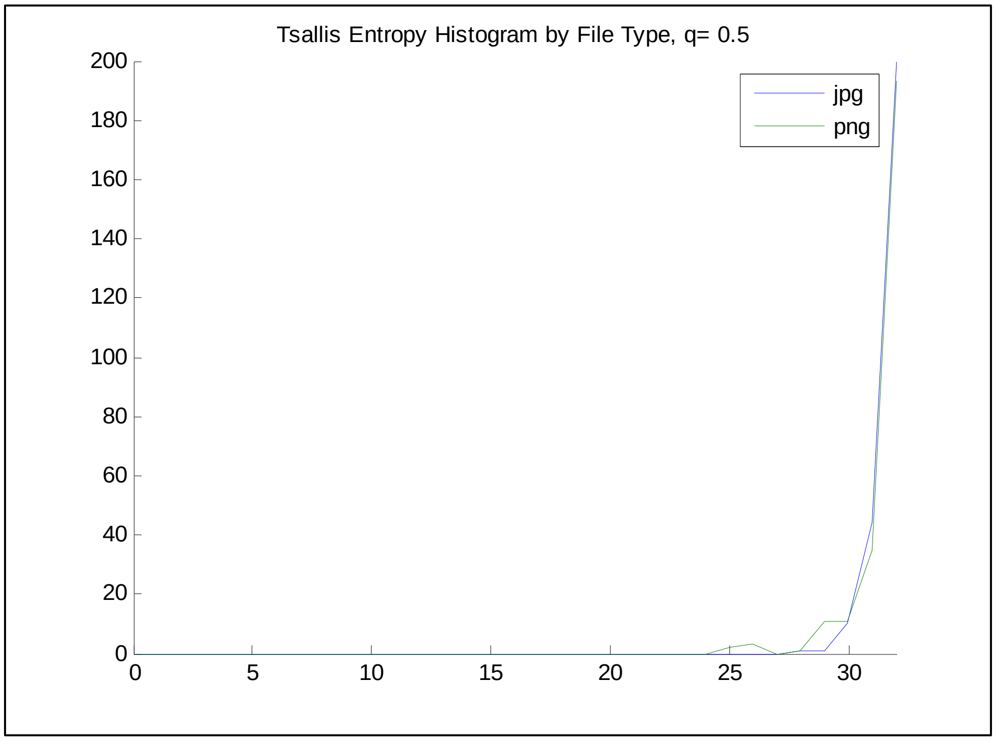


Figure 7 - Tsallis q-Entropy PNG & JPEG q = 0.5

2.7 Segment to Segment Features

While processing certain types of files, primarily Microsoft Office content, we noticed some significant features (Figure 8) occurring in almost all the files tested.

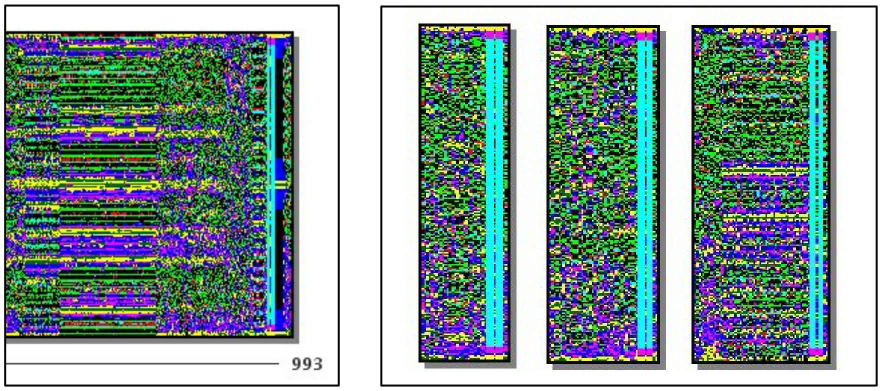


Figure 8 – Microsoft Office File-print Features

These features led us to believe that more information about MIME-type could be extracted from the DEF process if only certain segments of the files were taken into consideration, namely their heads and tails. The following results were generated in the same way as the first set but only the segments of the file specified were processed.

	BMP	**DOC**	**XLS**	**PPT**	**PNG**	**GIF**	**JPG**	**PDF**
BMP	**97.80**	0.80	0.40	1.00	0.00	0.00	0.00	0.00
DOC	1.00	**68.20**	9.80	13.80	0.20	0.60	1.20	5.20
XLS	3.61	8.02	**72.14**	8.22	0.80	0.80	0.00	6.41
PPT	1.20	10.44	8.63	**69.88**	0.80	0.20	2.01	6.83
PNG	0.60	0.00	0.00	1.00	**39.76**	23.90	26.71	8.03
GIF	0.20	0.00	0.00	0.00	10.71	**61.82**	21.41	5.86
JPG	0.80	0.20	1.40	0.60	4.41	22.24	**66.13**	4.21
PDF	1.20	0.80	1.60	5.40	2.40	2.60	8.20	**77.80**

Table 3 - Confusion Matrix, Last 10% of Files

	Office	CompImg	Other
DOC	**91.80**	2.00	6.20
XLS	**88.38**	1.60	6.41
PPT	**88.95**	3.01	8.04
PNG	1.00	**90.37**	8.63
GIF	0.00	**93.94**	6.06
JPG	2.20	**92.79**	5.01
PDF	7.80	13.20	**79.00**
BMP	2.20	0.00	**97.80**

Table 4 - Confusion Matrix, Last 10% Simplified

	BMP	DOC	XLS	PPT	PNG	GIF	JPG	PDF
BMP	**96.80**	0.80	0.60	0.80	0.00	0.80	0.00	0.20
DOC	0.80	**64.60**	11.80	16.40	0.40	0.20	0.40	5.40
XLS	2.81	6.21	**75.55**	8.22	0.40	2.20	0.40	4.21
PPT	2.01	3.01	13.45	**74.30**	0.00	0.20	1.00	6.02
PNG	0.80	0.20	0.20	0.60	**21.69**	32.33	35.74	8.43
GIF	0.61	0.00	0.20	0.40	2.42	**46.26**	42.22	7.88
JPG	1.60	0.40	0.80	0.40	5.81	19.44	**65.13**	6.41
PDF	3.00	0.20	2.40	2.20	4.20	3.80	2.40	**81.80**

Table 5 - Confusion Matrix, Last 5% of Files

	Office	CompImg	Other
DOC	**92.80**	1.00	6.20
XLS	**89.98**	3.00	7.02
PPT	**90.76**	1.20	8.03
PNG	1.00	**89.76**	9.23
GIF	0.60	**90.90**	8.49
JPG	1.60	**90.38**	8.01
PDF	4.80	10.40	**84.80**
BMP	2.20	0.80	**97.00**

Table 6 - Confusion Matrix, Last 5% Simplified

	BMP	**DOC**	**XLS**	**PPT**	**PNG**	**GIF**	**JPG**	**PDF**
BMP	**95.80**	0.20	3.80	0.00	0.00	0.20	0.00	0.00
DOC	1.00	**86.00**	4.60	0.40	0.00	0.80	0.20	7.00
XLS	5.81	5.41	**83.57**	0.60	0.00	1.80	0.80	2.00
PPT	1.20	7.63	5.02	**65.26**	0.60	1.00	3.82	15.46
PNG	0.60	0.20	0.40	19.48	**41.57**	18.07	14.06	5.62
GIF	2.02	0.61	0.00	7.68	16.57	**53.74**	9.29	10.10
JPG	0.60	2.20	0.40	15.03	10.02	15.23	**45.29**	11.22
PDF	0.40	9.20	11.20	26.80	6.80	3.40	7.20	**35.00**

Table 7 - Confusion Matrix, First 10% of Files

	Office	**CompImg**	**Other**
DOC	**91.00**	1.00	8.00
XLS	**89.58**	2.60	7.81
PPT	**77.91**	5.42	16.66
PNG	20.08	**73.70**	6.22
GIF	8.29	**79.60**	12.12
JPG	17.63	**70.54**	11.82
PDF	47.20	17.40	**35.40**
BMP	4.00	0.20	**95.80**

Table 8 - Confusion Matrix, First 10% Simplified

	BMP	**DOC**	**XLS**	**PPT**	**PNG**	**GIF**	**JPG**	**PDF**
BMP	**97.40**	0.00	1.20	0.00	0.00	1.00	0.00	0.40
DOC	1.20	**84.20**	2.80	1.20	0.00	0.20	0.60	9.80
XLS	5.01	2.20	**81.96**	0.40	0.00	5.61	1.40	3.41
PPT	3.21	3.61	4.22	**68.47**	0.60	1.41	3.61	14.86
PNG	0.80	0.20	0.40	19.68	**45.78**	16.47	12.65	4.02
GIF	2.02	0.61	0.20	9.90	16.97	**52.12**	9.29	8.89
JPG	0.40	0.40	2.40	12.63	11.82	12.83	**49.30**	10.22
PDF	1.80	3.40	13.40	27.20	6.60	4.20	8.40	**35.00**

Table 9 - Confusion Matrix, First 5% of Files

	Office	**CompImg**	**Other**
DOC	**88.20**	0.80	11.00
XLS	**84.56**	7.01	8.42
PPT	**76.30**	5.62	18.07
PNG	20.28	**74.90**	4.82
GIF	10.71	**78.38**	10.91
JPG	15.43	**73.95**	10.62
PDF	44.00	19.20	**36.80**
BMP	1.20	1.00	**97.80**

Table 10 - Confusion Matrix, First 5% Simplified

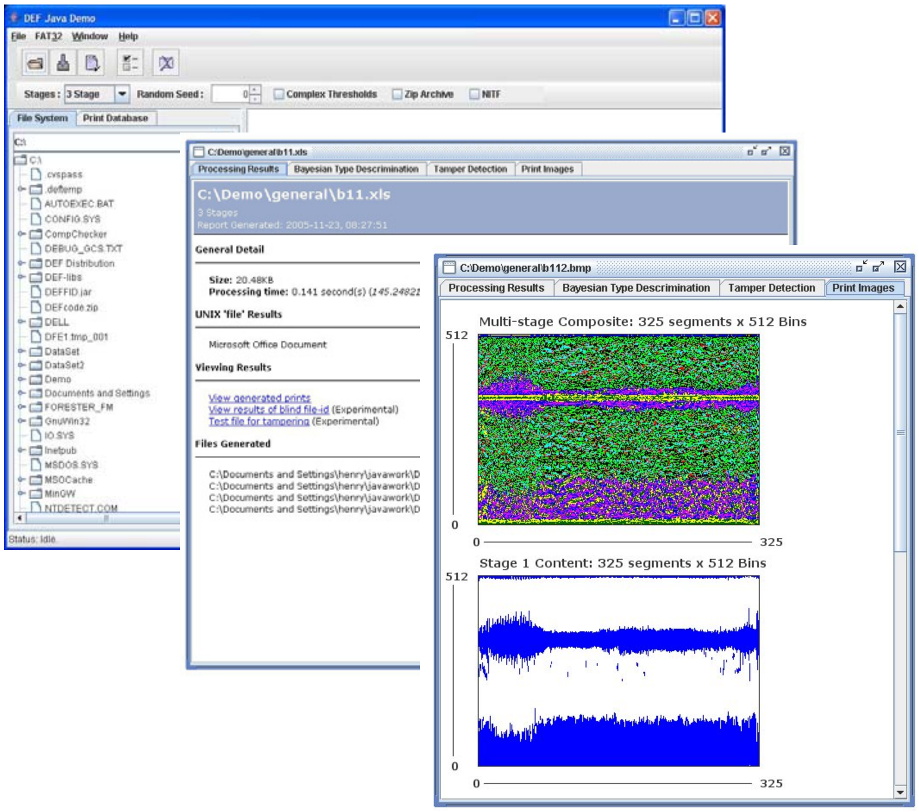
While this technique of looking at specific portions of the file does improve standings when looking at specific MIME-types, it does cause some extra confusion between the top level groupings of office files, compressed imagery and other files. This is not surprising when it is taken into account that the Bayesian classifier is at heart a statistical process which means it will benefit from more data segments.

2.8 Bayesian Networks

The goal of the initial Bayesian implementation was to develop a general purpose one step approach to the problem of MIME-type identification. Another Bayesian concept that could yield improvement in the DEF data stream identification process is networks. Currently we only use a simple one step Bayesian classifier but there may be effective ways to simplify or group results in one pass on the data and then process the data looking for more specific information with another classifier.

3. Implementation

The base implementation, extended under this effort, is an application written in the Java programming language (Figure 9). We standardized, in the early stages of development, on version 1.4.2 of the Java runtime, but later moved to version 1.5.0 for the improvements in Swing and HTML-support. Goals for the original effort included the development of a base application that would be portable, while providing an efficient demonstration of the actual and potential capabilities of the DEF algorithm.



C:\Demo\general\b11.xls
3 Stages
Report Generated: 2005-11-23, 08:27:51

General Detail

Size: 20.48KB
Processing time: 0.141 second(s) (*145.24821*

UNIX 'file' Results

Microsoft Office Document

Viewing Results

View generated prints
View results of blind file-id (Experimental)
Test file for tampering (Experimental)

Files Generated

C:\Documents and Settings\henry\javawork\D
C:\Documents and Settings\henry\javawork\D
C:\Documents and Settings\henry\javawork\D
C:\Documents and Settings\henry\javawork\D

Figure 9 - The DEF Demonstration Application

The extension effort focused on enhancing functionality by including the enhancement of the viewing DEF file-prints and related outputs, more control of algorithm parameters, schemes under which files can be protected and verified, MIME-type identification and enhancing the quality of the results from the original effort.

The application is driven by the “file” menu option and the file explorer pane in the main user interface via contextual menu options. After files are processed the results of that operation are available via the print database tree view which lists the files name, number of stages in processing and the date processed.

The following sections of this paper describe the individual components of the application and their uses.

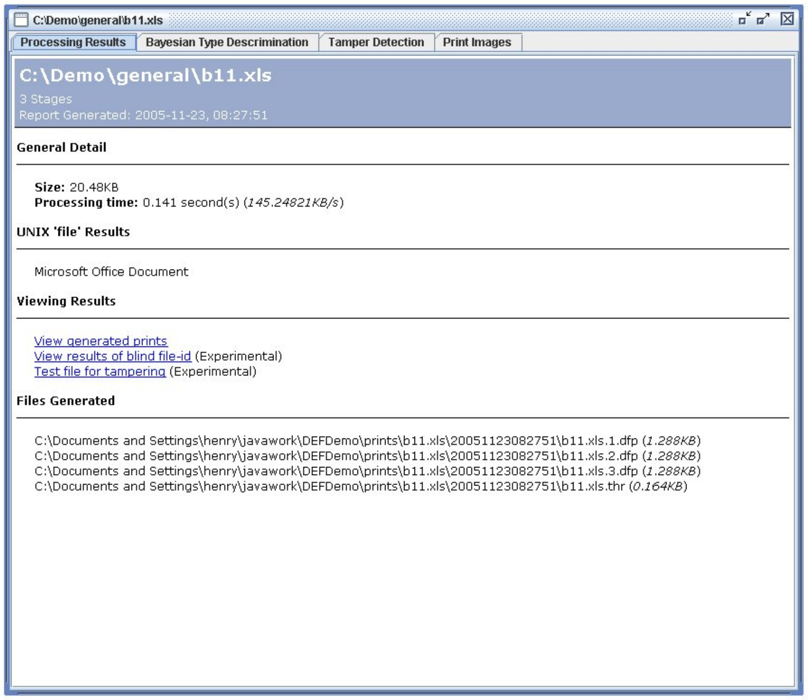
3.1 General Processing Reports

The general processing details for the DEF process are presented via one of three general processing reports. Which report is provided is decided by the number of files being processed and configurable parameters for the number of files at which batch modes need.

3.2 Individual File Report

When the user processes a file under default circumstances they are presented with the general processing report shown in Figure 10. This report supplies the user with general processing information including:

- Processed file name
- Number of stages in processing
- Date and time
- Size of processed file
- Time spent during DEF processing operation
- Optional UNIX 'file' results for the target file
- Links to detailed information
- Details on each DEF output



C:\Demo\general\b11.xls
3 Stages
Report Generated: 2005-11-23, 08:27:51

General Detail

Size: 20.48KB
Processing time: 0.141 second(s) (*145.24821KB/s*)

UNIX 'file' Results

Microsoft Office Document

Viewing Results

View generated prints
View results of blind file-id (Experimental)
Test file for tampering (Experimental)

Files Generated

```
C:\Documents and Settings\henry\javawork\DEFDemo\prints\b11.xls\20051123082751\b11.xls.1.dfp (1.288KB)
C:\Documents and Settings\henry\javawork\DEFDemo\prints\b11.xls\20051123082751\b11.xls.2.dfp (1.288KB)
C:\Documents and Settings\henry\javawork\DEFDemo\prints\b11.xls\20051123082751\b11.xls.3.dfp (1.288KB)
C:\Documents and Settings\henry\javawork\DEFDemo\prints\b11.xls\20051123082751\b11.xls.thr (0.164KB)
```

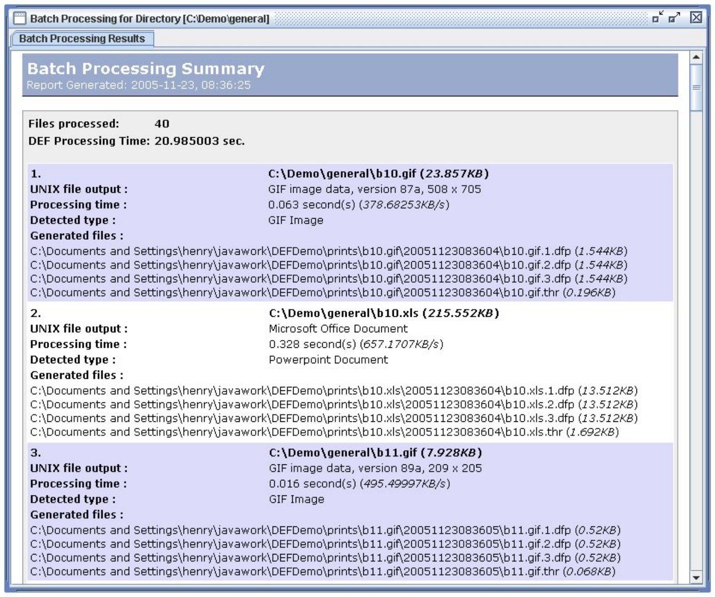
Figure 10 - General Processing Report

The UNIX file results will only be available if the application is properly configured and a compatible version of that program is installed on the target system. The GNU version of this application is available for UNIX, Linux and Windows systems and is compatible with this application.

3.3 Batch Mode Level 1

The level 1 batch mode provides general level information about the files but does not include access to the detailed information about the process such as the Bayesian classifier information or print image view (Figure 11). The following information is supplied in a level 1 batch report:

- Date and time of processing
- File name and size
- DEF processing time
- UNIX ‘file’ results
- DEF/Bayes MIME-type ID results
- DEF outputs details



```
Batch Processing Summary
Report Generated: 2005-11-23, 08:36:25

Files processed:        40
DEF Processing Time: 20.985003 sec.

1.                                  C:\Demo\general\b10.gif (23.857KB)
UNIX file output :                  GIF image data, version 87a, 508 x 705
Processing time :                   0.063 second(s) (378.68253KB/s)
Detected type :                     GIF Image
Generated files :
C:\Documents and Settings\henry\javawork\DEFDemo\prints\b10.gif\20051123083604\b10.gif.1.dfp (1.544KB)
C:\Documents and Settings\henry\javawork\DEFDemo\prints\b10.gif\20051123083604\b10.gif.2.dfp (1.544KB)
C:\Documents and Settings\henry\javawork\DEFDemo\prints\b10.gif\20051123083604\b10.gif.3.dfp (1.544KB)
C:\Documents and Settings\henry\javawork\DEFDemo\prints\b10.gif\20051123083604\b10.gif.thr (0.196KB)
2.                                  C:\Demo\general\b10.xls (215.552KB)
UNIX file output :                  Microsoft Office Document
Processing time :                   0.328 second(s) (657.1707KB/s)
Detected type :                     Powerpoint Document
Generated files :
C:\Documents and Settings\henry\javawork\DEFDemo\prints\b10.xls\20051123083604\b10.xls.1.dfp (13.512KB)
C:\Documents and Settings\henry\javawork\DEFDemo\prints\b10.xls\20051123083604\b10.xls.2.dfp (13.512KB)
C:\Documents and Settings\henry\javawork\DEFDemo\prints\b10.xls\20051123083604\b10.xls.3.dfp (13.512KB)
C:\Documents and Settings\henry\javawork\DEFDemo\prints\b10.xls\20051123083604\b10.xls.thr (1.692KB)
3.                                  C:\Demo\general\b11.gif (7.928KB)
UNIX file output :                  GIF image data, version 89a, 209 x 205
Processing time :                   0.016 second(s) (495.49997KB/s)
Detected type :                     GIF Image
Generated files :
C:\Documents and Settings\henry\javawork\DEFDemo\prints\b11.gif\20051123083605\b11.gif.1.dfp (0.52KB)
C:\Documents and Settings\henry\javawork\DEFDemo\prints\b11.gif\20051123083605\b11.gif.2.dfp (0.52KB)
C:\Documents and Settings\henry\javawork\DEFDemo\prints\b11.gif\20051123083605\b11.gif.3.dfp (0.52KB)
C:\Documents and Settings\henry\javawork\DEFDemo\prints\b11.gif\20051123083605\b11.gif.thr (0.068KB)
```

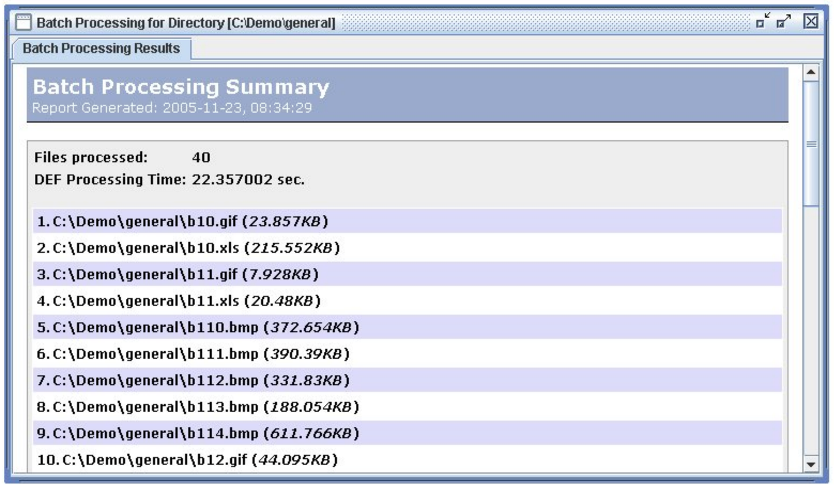
Figure 11 - Batch Mode, Level 1

This mode is provided for those instances in which the user wants some detailed information but generation of print images and details for every file being processed is not practical.

3.4 Batch Mode Level 2

The level 2 batch mode provides information only general details about the files generated by the process (Figure 12). The following information is supplied in a level 2 batch report:

- Date and time of processing
- Number of files processed
- DEF processing time
- Processed files names and sizes



Batch Processing for Directory [C:\Demo\general]

Batch Processing Results

Batch Processing Summary
Report Generated: 2005-11-23, 08:34:29

```
Files processed:       40
DEF Processing Time: 22.357002 sec.

1. C:\Demo\general\b10.gif (23.857KB)
2. C:\Demo\general\b10.xls (215.552KB)
3. C:\Demo\general\b11.gif (7.928KB)
4. C:\Demo\general\b11.xls (20.48KB)
5. C:\Demo\general\b110.bmp (372.654KB)
6. C:\Demo\general\b111.bmp (390.39KB)
7. C:\Demo\general\b112.bmp (331.83KB)
8. C:\Demo\general\b113.bmp (188.054KB)
9. C:\Demo\general\b114.bmp (611.766KB)
10. C:\Demo\general\b12.gif (44.095KB)
```

Figure 12 - Batch Mode, Level 2

This report is supplied to provide the user with the ability to process large numbers of files with the intent of using the outputs at a later date. The outputs can be examined using the print database tree view.

3.5 DEF Print View

The DEF print view gives the user a visualization of the outputs of the DEF algorithm (Figure 13). These images give the user the ability to look for visual information in the prints and to look for patterns between files and types. The prints for all stages and a composite print image are provided to the user.

Each image represents the outputs of the ABC detector which is half of the transform size in the Y dimension, representing frequency content, and the number of processing blocks in the X dimension, representing each processed segment of the source data stream.

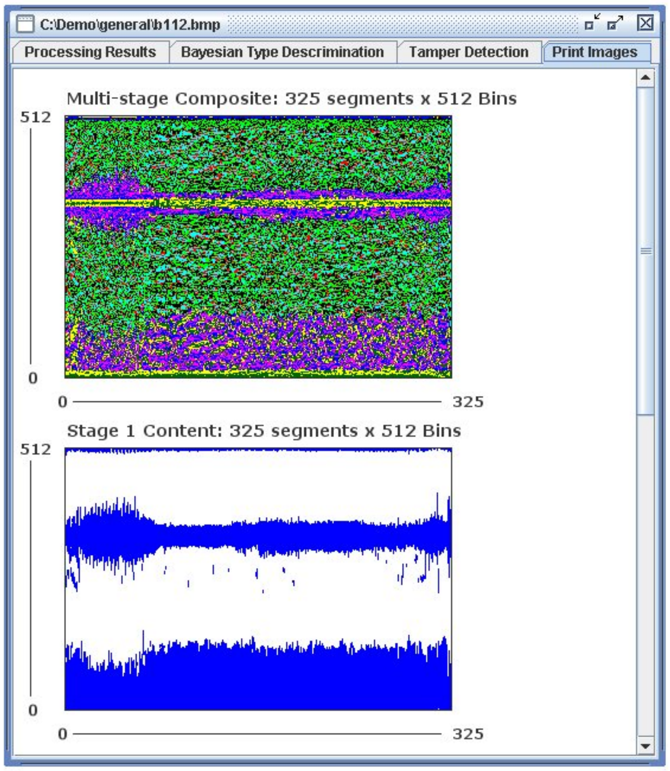
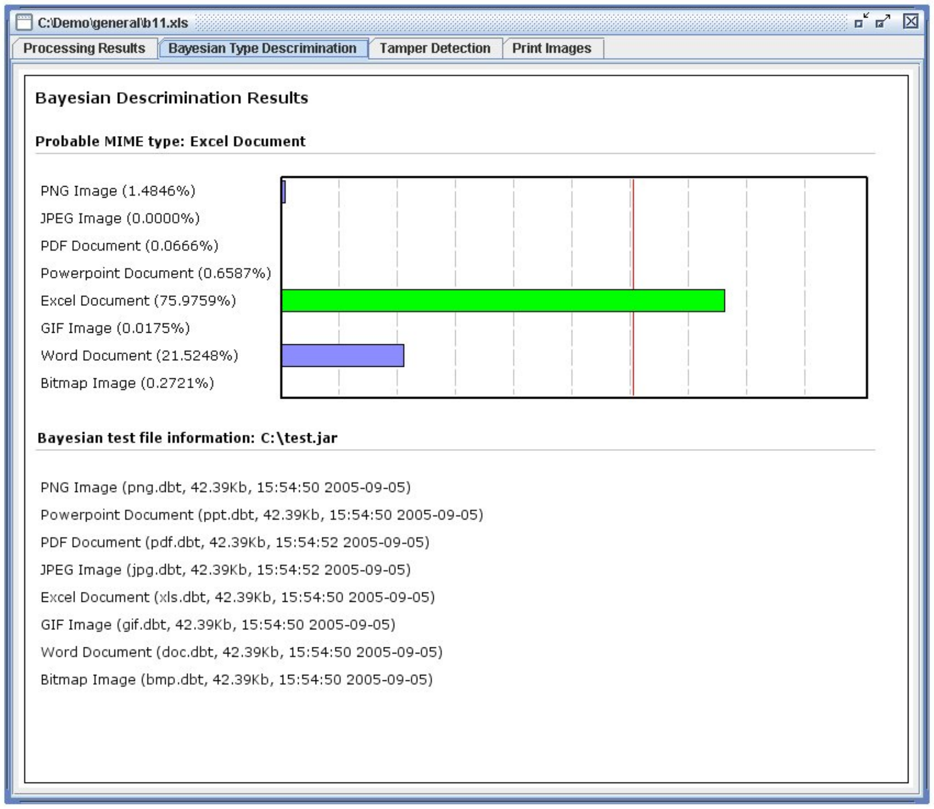

C:\Demo\general\b112.bmp
Processing Results
Bayesian Type Descrimination
Tamper Detection
Print Images
Multi-stage Composite: 325 segments x 512 Bins
512
0
0
325
Stage 1 Content: 325 segments x 512 Bins
512
0
0
325

Figure 13 - The DEF Print View

3.6 Bayesian Type Discrimination Report

The type discrimination report, shown in Figure 14, provides the user with the results of the Bayesian discrimination process. The file is processed and DEF outputs are created, DEF features are generated. These features are used to execute lookups into each file-types histogram and the results of the Bayesian calculation are displayed as a bar chart. The DEF detected type will have a green bar unless it did not meet the threshold for the minimum acceptable percentage of certainty, which will subsequently appear in red.



Bayesian Descrimination Results

Probable MIME type: Excel Document

PNG Image (1.4846%)
JPEG Image (0.0000%)
PDF Document (0.0666%)
Powerpoint Document (0.6587%)
Excel Document (75.9759%)
GIF Image (0.0175%)
Word Document (21.5248%)
Bitmap Image (0.2721%)

Bayesian test file information: C:\test.jar

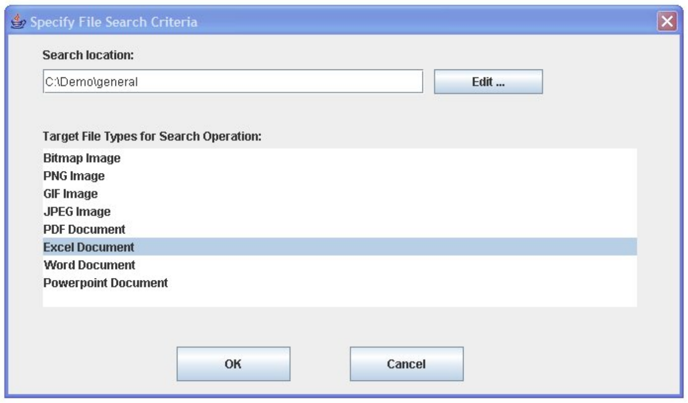
PNG Image (png.dbt, 42.39Kb, 15:54:50 2005-09-05)
Powerpoint Document (ppt.dbt, 42.39Kb, 15:54:50 2005-09-05)
PDF Document (pdf.dbt, 42.39Kb, 15:54:52 2005-09-05)
JPEG Image (jpg.dbt, 42.39Kb, 15:54:52 2005-09-05)
Excel Document (xls.dbt, 42.39Kb, 15:54:50 2005-09-05)
GIF Image (gif.dbt, 42.39Kb, 15:54:50 2005-09-05)
Word Document (doc.dbt, 42.39Kb, 15:54:50 2005-09-05)
Bitmap Image (bmp.dbt, 42.39Kb, 15:54:50 2005-09-05)

Figure 14 - The Bayesian Type Discrimination Report

This report also provides the user with details about the discriminator being used including the archive and comprising test files, their sizes and a time stamp providing their creation times.

3.7 MIME-Type Search

The MIME-type search capability in the demonstration application is meant to provide a basic application of the DEF Bayesian classifier. The user selects a group of files and the MIME-types that they wish to find with dialog shown in Figure 15. The selected files are processed with the DEF algorithm and the Bayesian classifier is applied to those outputs. If a file is classified as one of the requested MIME-types, it is reported to the user using the report shown in Figure 16.



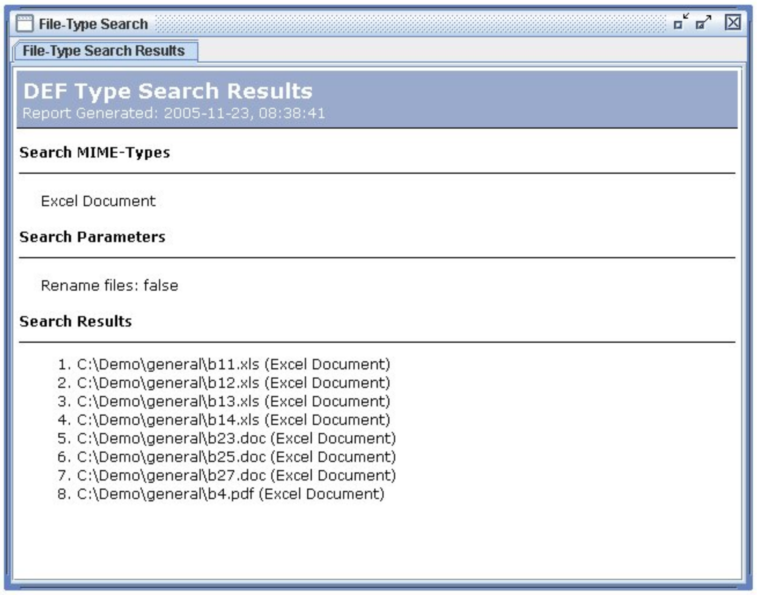
Search location:

C:\Demo\general

Target File Types for Search Operation:

Bitmap Image
PNG Image
GIF Image
JPEG Image
PDF Document
Excel Document
Word Document
Powerpoint Document

Figure 15 - MIME-Type Search Configuration



DEF Type Search Results
Report Generated: 2005-11-23, 08:38:41

Search MIME-Types

Excel Document

Search Parameters

Rename files: false

Search Results

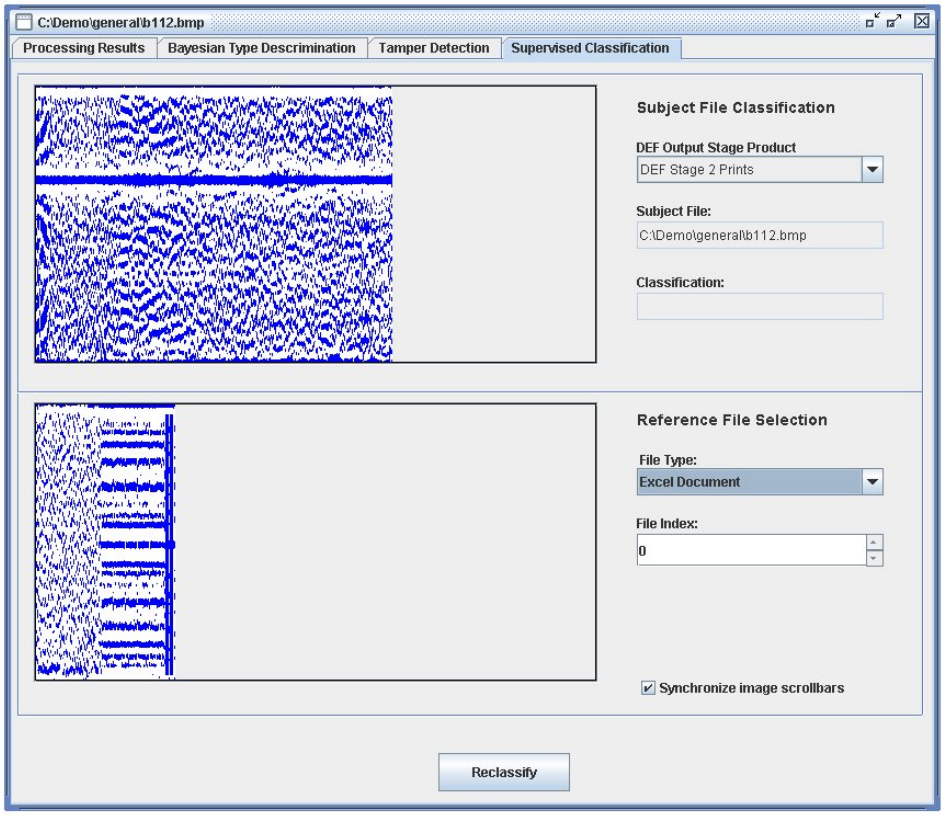
```
1. C:\Demo\general\b11.xls (Excel Document)
2. C:\Demo\general\b12.xls (Excel Document)
3. C:\Demo\general\b13.xls (Excel Document)
4. C:\Demo\general\b14.xls (Excel Document)
5. C:\Demo\general\b23.doc (Excel Document)
6. C:\Demo\general\b25.doc (Excel Document)
7. C:\Demo\general\b27.doc (Excel Document)
8. C:\Demo\general\b4.pdf (Excel Document)
```

Figure 16 - MIME-Type Search Results

3.8 Supervised Classification

The supervised classification mode (Figure 17) was provided as a method for combining user expertise and interpretation with the Bayesian classifier described in previous sections.

The user selects a file and requests supervised classification, DEF outputs are generated for the file and then the user is provided with a dialog in which they can do a direct comparison between those outputs and a group of previously generated prints provided to represent the known MIME-types. The user makes a visual inspection of this information and if they decide they can exclude a particular MIME-type from consideration they can remove it as a hypothesis for the classifier.



Subject File Classification

DEF Output Stage Product: DEF Stage 2 Prints

Subject File: C:\Demo\general\b112.bmp

Classification:

Reference File Selection

File Type: Excel Document

File Index: 0

☑ Synchronize image scrollbars

Figure 17 - Supervised Classification Dialog

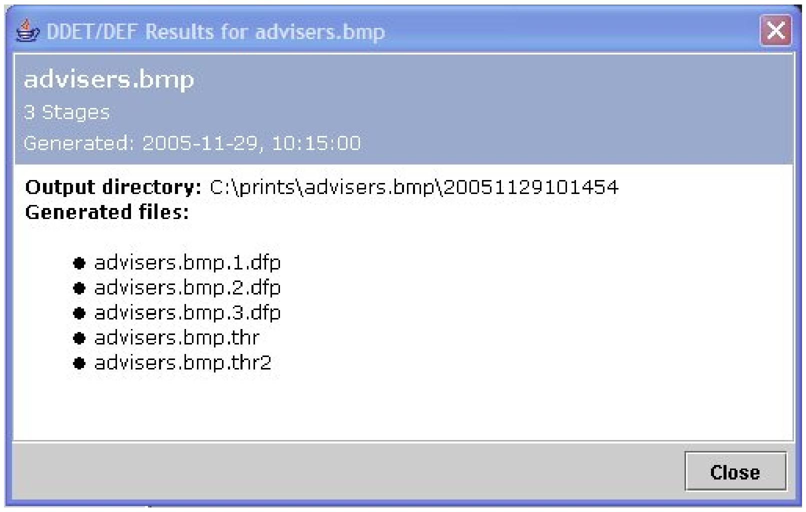
3.9 Raw Disk I/O Demonstration

A raw disk I/O mode was provided in the DEF demonstration application to provide some insight into potential uses of the DEF algorithm in a data forensics and recovery context. If the demonstration application is running on the Windows operating system and there are FAT32 file-systems available the user can scan that partition for deleted files of specific MIME-types and then recover those files for use.

This function makes use of the MIME-type search functionality to make its evaluations. The module itself is implemented as a C++ dynamic link library compiled with Microsoft Visual Studio 6.0 and makes use of the Java JNI API. The use of C++ was required to open and read the disk in a raw fashion. Windows was chosen because it is the most common platform while FAT32 was chosen for its simplicity.

3.10 PAR GeoView Plug-in Implementation

GeoView, a PAR GeoWare product, is an extensible platform for manipulating and analyzing imagery. As a part of the extension effort we wrote a plug-in for that application providing the tamper detection capabilities of the DEF algorithm. The user is allowed to protect files using the DEF algorithm. The output of this process is shown in Figure 18.



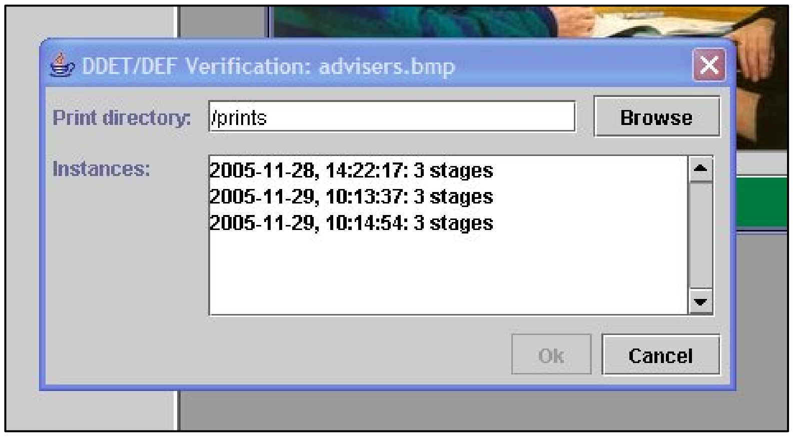
advisers.bmp
3 Stages
Generated: 2005-11-29, 10:15:00

Output directory: C:\prints\advisers.bmp\20051129101454
Generated files:

- advisers.bmp.1.dfp
- advisers.bmp.2.dfp
- advisers.bmp.3.dfp
- advisers.bmp.thr
- advisers.bmp.thr2

Figure 18 - PAR GeoView Plug-in DEF Report

Once a file is processed the user can verify that file against previously recorded instances, providing a simple protection scheme (Figure 19) for images supported by the GeoView application.



Print directory: /prints

Instances:
2005-11-28, 14:22:17: 3 stages
2005-11-29, 10:13:37: 3 stages
2005-11-29, 10:14:54: 3 stages

Figure 19 - PAR GeoView Plug-in DEF Verification

4. Conclusions and Recommended Future Efforts

In this effort, we demonstrated the DEF algorithm, its general purpose application towards MIME-type file identification, as well as its use for data tamper protection.

The DEF tamper detection outputs have a unique combination of compressed size and localization capabilities that make it a very attractive concept for further study. If further developed, these capabilities could have applications in many of the areas contemporary hashes and checksums currently occupy, while providing capabilities that those algorithms can never hope to accomplish for tamper localization.

The Java implementation of DEF and the implementation of the GeoView application plug-in show the portable and encapsulated nature of the algorithm. This demonstrates some of the potential for this algorithm to be embedded into other contexts and applications.

New file formats are being created faster than you can count them, and techniques that hope to be able to detect these types, without depending on specific markers or magic numbers in those files, will have to specialize and focus to be accurate. The DEF algorithm provides a source of data that can be shaped into useful features that can help solve this problem. The demonstration software application developed under this effort is a simplified look at what can be done with these DEF features and a Bayesian classifier. Even in this basic context it proves itself to be fairly competent when investigating groups of files such as Microsoft Office documents and compressed imagery files.

More specific tests could be generated to help separate those larger groups into more specific ones, more Bayesian or other probabilistic techniques could be combined and a network could be developed that has the potential for being a very powerful tool in solving the problems of identifying unknown data streams.

5. References:

[1] "Detector and Extractor of Fileprints (DEF) Prototype," AFRL-IF-RS-TR-2004-136, Final Technical Report, May 2004.

[2] U.S. Patent 5,257,211 "Adjustable Bandwidth Concept Signal Energy Detector," October 1993.

[3] Patent Pending; see http://www.uspto.gov Publication Number US/2004 0078574-A1

[4] Shannon, C.E., "A Mathematical Theory of Communication", The Bell System Technical Journal, Vol. 27, July, October 1948

[5] Tsallis,C. and MGell-Mann, M. (ed.), "Nonextensive Entropy", Oxford University Press, 2004.